ELEC6233
Digital System Synthesis

4. Binary Decision Diagrams and SAT solvers

Methods developed in 1990s – local search

- Able to perform local optimisation:
  - Starting point:
    - a certain variable assignment
  - Cost function (Penalty function):
    - number of unsatisfied clauses in the Boolean function
  - Basic procedure:
    - Move to an adjacent point in the Boolean space by flipping one variable assignment, recalculate the cost
- Local minima:
  - Heuristically accept moves that worsen the cost function to exit from local minima – this is where BDDs offer great potential!
- Such solvers are typically incomplete
  - i.e. cannot prove unsatisfiability

Further developments

- 1994: Hannibal
  - 3000 variables
- 1996: Stalmarck’s algorithm
  - 1000 variables
- 1996: GRASP
  - Conflict driven learning and non-chronological backtracking
  - Practical SAT problems in high-level synthesis can be solved in reasonable time
    - 1000 variables
- 1997: RelSAT – also proposed conflict driven learning

Binary Decision Diagrams – efficient Boolean space search

- Key paper:
- Idea:
  - Store the Boolean function in a Directed Acyclic Graph (DAG) representation.
  - Compacted form of the binary decision tree.
- Reduction rules to manipulate the graph
- Great potential for exploiting heuristics in Boolean search.
- In 1992 GSAT tool reported:
  - Efficient search in 300 dimensions

1994: Hannibal
- 3000 variables

1996: Stalmarck’s algorithm
- 1000 variables

1996: GRASP
- Conflict driven learning and non-chronological backtracking
- Practical SAT problems in high-level synthesis can be solved in reasonable time
  - 1000 variables

1997: RelSAT – also proposed conflict driven learning
Conflict Driven Learning and Non-Chronological Backtracking example

\[ a + d \]
\[ a + c + h \]
\[ a + h + l \]
\[ b + k \]
\[ g + c + i \]
\[ g + h + i \]
\[ g + h + j \]
\[ g + j + l \]

\[ a = 0 \]

Conflict Driven Learning and Non-Chronological Backtracking example

\[ a + d \]
\[ a + c + h \]
\[ a + h + l \]
\[ b + k \]
\[ g + c + i \]
\[ g + h + i \]
\[ g + h + j \]
\[ g + j + l \]
\[ c = 1 \]

\[ a = 0, d = 1 \text{(implied)} \]

Conflict Driven Learning and Non-Chronological Backtracking example

\[ a + d \]
\[ a + c + h \]
\[ a + h + l \]
\[ b + k \]
\[ g + c + i \]
\[ g + h + i \]
\[ g + h + j \]
\[ g + j + l \]

\[ a = 0, d = 1 \text{(implied)} \]

Conflict Driven Learning and Non-Chronological Backtracking example

\[ a + d \]
\[ a + c + h \]
\[ a + h + l \]
\[ b + k \]
\[ g + c + i \]
\[ g + h + i \]
\[ g + h + j \]
\[ g + j + l \]
\[ c = 1, h = 0 \text{(implied)} \]
Conflict Driven Learning and Non-Chronological Backtracking example

\[
\begin{align*}
\text{Conflict } i=0, j &= \text{ implied} \\
\text{Backtrack to the decision level of variable } c \text{ with implication } g=0
\end{align*}
\]

Conflict-Driven Learning - advantages

- Learned clause is useful forever!
- Useful in generating future conflict clauses
- Can restart, i.e. abandon the current search tree and reconstruct a new one
  - Adds to robustness in the solver
  - The clauses learned before the restart are still in the Boolean function after the restart and can help pruning the search space
Conflict Driven Learning and Non-Chronological Backtracking example

\[\begin{align*}
a + d \\
a + c + h \\
a + h + l \\
b + h \\
g + c + h \\
g + h + f \\
g + h + j \\
g + j + l \\
n + c + h
\end{align*}\]

SAT assignment: \(a=0, b=0, c=0, d=1, g=1, h=1, l=1\)

2001: CHAFF – very efficient SAT solver

- One to two orders of magnitude faster than other SAT solvers
- Widely Used:
  - Formal verification
  - Hardware and software
  - NuSMV – Symbolic Verification toolset
  - Automatic theorem provers
  - Alloy – Software Model Analyzer at M.I.T.
  - halfWay – Refutation-based first-order logic theorem prover
  - Several industrial users – Intel, IBM, Microsoft, ...

After adoption of Conflict-Driven Learning, SAT became practical

- Conflict driven learning greatly increases the capacity of SAT solvers
- Realistic high-level synthesis applications became plausible
  - Nowadays thousands and even millions of variables are handled
  - Typical applications in Electronic Design Automation that can make use of SAT
    - Formal circuit verification without simulation
    - FPGA routing
    - Scheduling tasks
    - Many others...
- Research direction changes towards more efficient implementations

Large example attempted by CHAFF

- Industrial processor verification reported
  - 14 cycle behavior
- Statistics
  - 1 million variables
  - 10 million literals initially
  - 200 million literals including added clauses
  - 30 million literals finally
  - 4 million clauses (initially)
  - 200K clauses added
  - 1.5 million decisions
  - 3 hours run time
CHAFF Approach

- Make the core operations fast
  - most time-consuming parts:
    - Boolean Constraint Propagation (BCP – more on this to follow) and Decision Trees
- Emphasis on coding efficiency and elegance
- Emphasis on optimization of data cache behaviour
- Emphasis on good search space pruning, i.e. conflict resolution and learning

CHAFF challenges: large (in-memory) database, CPU intensive search

Boolean Constraint Propagation

- Boolean Constraint Propagation (BCP) == Unit Propagation (UP) == One-literal Rule (OLR)
- BCP is based on unit clauses, i.e. clauses that are composed of a single literal
- If a set of clauses contains the unit clause U, the other clauses are simplified by the iterative application of the following two rules:
  1. every clause (other than clause U itself) containing U is removed
  2. in every clause that contains the negation of U: \( \bar{U} \), the literal \( \bar{U} \) is removed
- The application of these two rules leads to a new, simpler set of clauses, that is equivalent to the old one.

BCP example

\[ F = a(a + b)(\bar{a} + c)(\bar{e} + d) \]

1. since \((a + b)\) contains \(a\), this clause can be removed
2. since \((\bar{a} + c)\) contains the negation of \(a\), \(\bar{a}\) can be removed from the clause

Hence: \( F = ac(\bar{e} + d) \)

3. since \((\bar{e} + d)\) contains the negation of \(c\), \(\bar{e}\) can be removed from the clause

Hence: \( F = acd \)

Exercise: prove by some method, algebra, truth table or K-map, that the above three forms are equivalent.

Many variants of BCP were proposed, e.g. SATO


- The idea:
  - Each clause has a head pointer and a tail pointer.
  - All literals in a clause before the head pointer and after the tail pointer have been assigned false.
  - SATO invariant: If a clause can become SAT via any sequence of assignments, then this sequence will include an assignment to one of the literals pointed to by the head/tail pointer.
Decision Heuristics – Common Sense

- DLIS (Dynamic Largest Individual Sum) is a relatively simple dynamic decision heuristic
  - Simple and intuitive: At each decision simply choose the assignment that satisfies the most unsatisfied clauses.
  - However, considerable work is required:
    - Must touch each clause that contains a literal that has been set to true. Often restricted to initial (not learned) clauses.
    - Maintain “SAT” counters for each clause
    - When counters transition 0→1, update rankings.
    - Need to reverse the process for unassignment.
  - The total effort required for this and similar decision heuristics may be significantly more than the basic BCP algorithm.
- Look ahead SAT algorithms even more CPU intensive, GPUs have been used recently:

How to verify a SAT Solver?

- If it claims the instance is satisfiable, it is easy to check the claim.
  - But how about unsatisfiability claims?
- An unsatisfactory search process is not necessarily a proof of unsatisfiability
- Need an independent check for SAT claims
- Checker must be automatic
  - Must be able to work with current state-of-the-art SAT solvers
- The SAT solver dumps a trace (on disk) during the solving process from which a resolution graph can be derived
- A third party checker constructs the empty clause by resolution using the trace

BerkMin SAT solver – Decision Making Heuristics

- Identify the most recently learned clause which is unsatisfied
- Pick most active variable in this clause to branch on
- Variable activities
  - updated during conflict analysis
  - decay periodically
- If all learnt conflict clauses are satisfied, choose a variable using a global heuristic
- Increased emphasis on “locality” of decisions

Extracting an unsatisfiable core from a bigger unsatisfiable logic problem

- Extract a small subset of unsatisfiable clauses from an unsatisfiable SAT instance
- Motivation:
  - Debugging and redesign: SAT instances are often generated from real world applications with certain expected results:
    - If the expected result is unsatisfiable, but the instance is satisfiable, then the solution is a “stimulus” or “input vector” or “counter-example” for debugging
      - Combinational Equivalence Checking
      - Bounded Model Checking
  - What if the expected result is satisfiable?
    - SAT Planning
    - FPGA Routing
  - Relaxing constraints in a design:
    - If several constraints make a certain property hold, are there any redundant constraints in the system that can be removed without violating the property?
• Rich history of emphasis on practical efficiency.
• Many successful applications reported.
• Need to account for computation cost in search space pruning.
• Need to match algorithms with underlying processor architectures.
  – GPUs, many core systems, and cloud clusters are used in recent years.
• Specific problem classes can benefit from specialized algorithms
  – Identification of problem classes?
  – Dynamically adapting heuristics?
• Research papers continue to be published — much room to learn and improve.

Summary of SAT